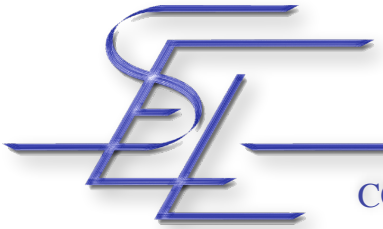
COMP-533

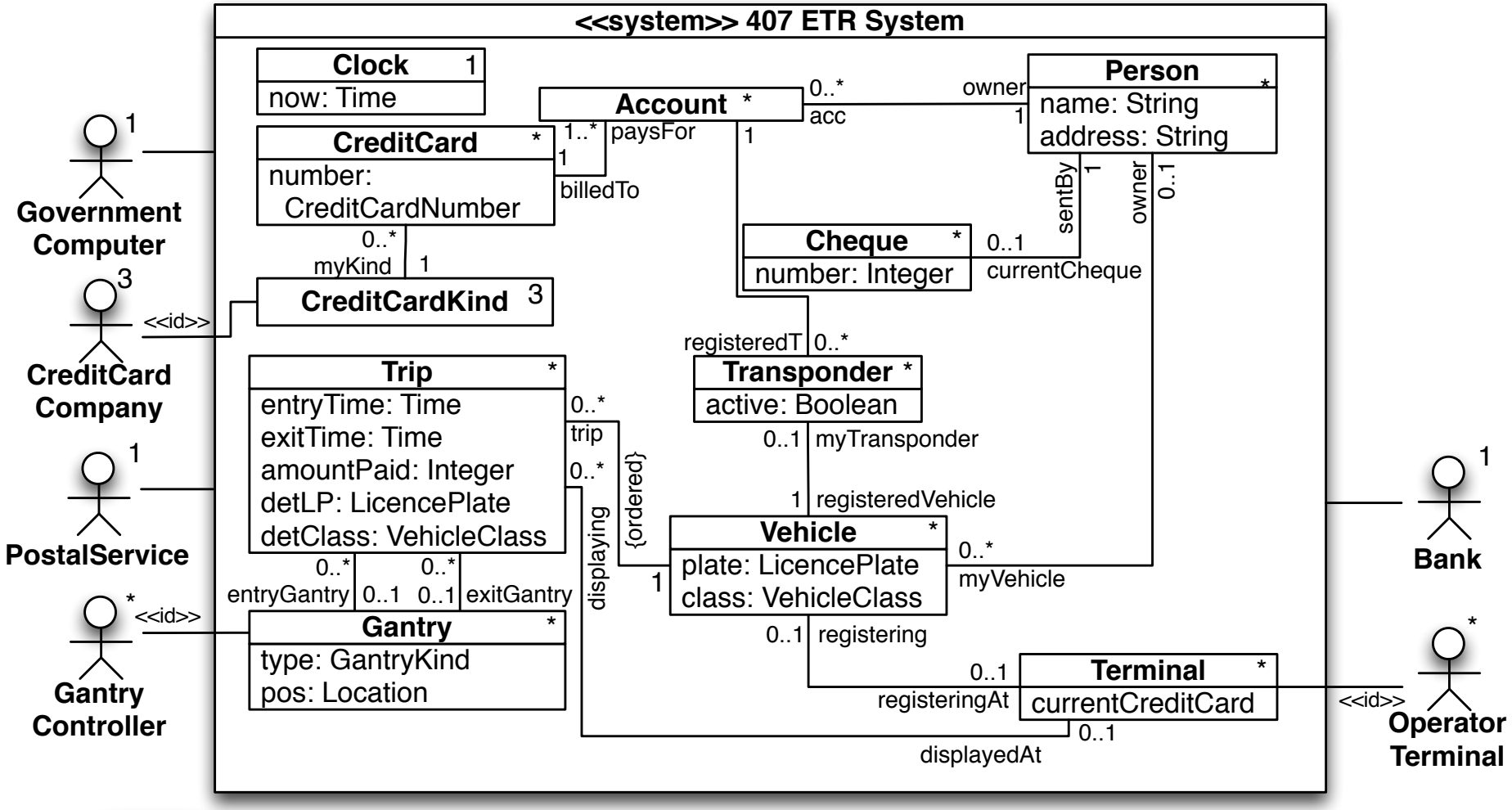
Solution 407 ETR 2012

Jörg Kienzle

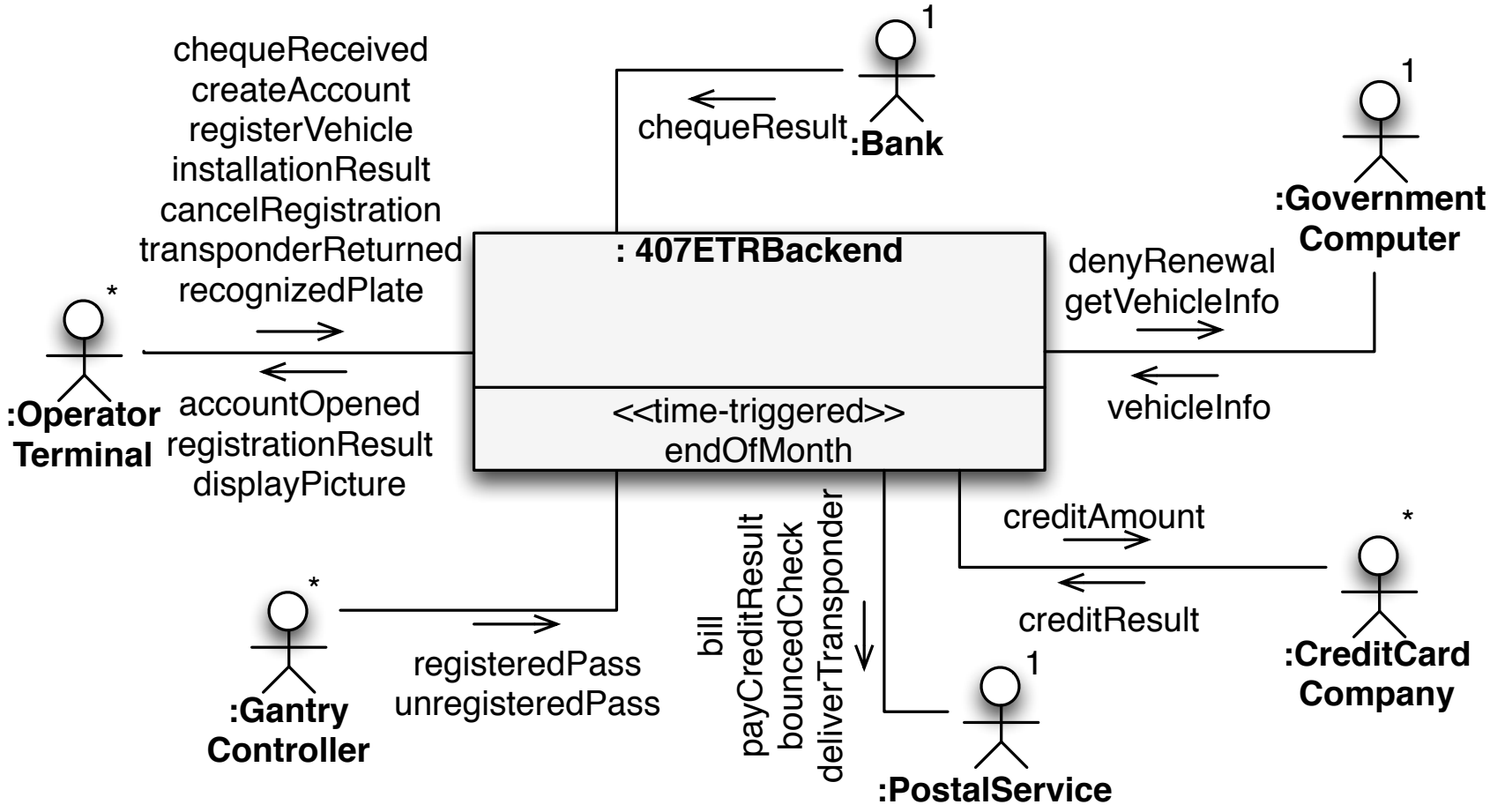


McGill

407 ETR Concept Model



407 ETR Environment Model



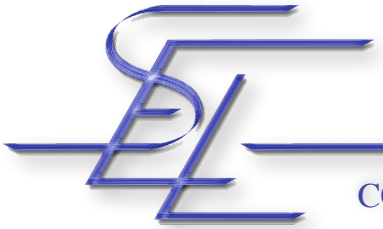
Operation CreateAccount

Operation: 407Backend::createAccount(name: String, address: String, cardnumber: CreditCardNumber);
Scope: Person, Account, CreditCard;
Messages: OperatorTerminal::{AccountOpened}
New:
  newAccount: Account; newPerson: Person; newCard: CreditCard
Alias:
  p: Person = **self**.person**@pre→any**(p | p.name = name **and** p.address = address);
Post:
  newAccount.**oclIsNew**() &
  if p.**oclIsUndefined**() **then**
    newPerson.**oclIsNew**(name ⇒ name, address ⇒ address) &
    newPerson.acc = newAccount
  else
    p.acc→**includes**(newAccount)
  endif &
  let card: CreditCard = **self**.creditCard→**any**(number = cardnumber) **in**
    if card.**oclIsUndefined**() **then**
      newCard.**oclIsNew**(number ⇒ cardnumber) &
      newCard.myKind = newCard.getKind() &
      newAccount.billedTo = newCard
    else
      newAccount.billedTo = card
    endif
  endlet &
  sender^accountOpened(newAccount)

McGill

Operation RegisterVehicle

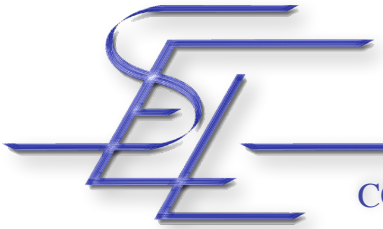
Operation: 407Backend::registerVehicle(a: Account, lp: LicencePlate, class: VehicleClass);
Scope: Account, Vehicle, Terminal, Transponder;
Messages: GovernmentComputer::{GetVehicleInfo}
New:
  newVehicle: Vehicle;
  newTransponder: Transponder;
Pre: **self**.vehicle→**select**(plate = lp)→**isEmpty**();
Post:
  newVehicle.**oclIsNew**((plate ⇒ lp, class ⇒ class) &
  newVehicle.registeringAt = **sender**.terminal &
  newVehicle.owner = a.owner &
  newTransponder.**oclIsNew**() &
   newVehicle.myTransponder = newTransponder &
  a.registeredT→**includes**(newTransponder) &
  self.governmentComputer^getVehicleInfo(lp)



McGill

Operation InstallationResult

Operation: 407Backend::installationResult(t: Transponder, outcome: Boolean);
Scope: Transponder, Vehicle;
Messages: PostalService::{DeliverTransponder}
New:
  newTransponder: Transponder;
Post:
 if outcome **then**
   t.active
 else
   newTransponder.**oclIsNew**() &
   t.registeredVehicle**@pre**.myTransponder = newTransponder &
   t.account**@pre**.registeredT→**includes**(newTransponder) &
   self.transponder→**excludes**(t) &
   self.postalService^deliverTransponder(t.account**@pre**.owner, newTransponder)
 endif



McGill

Operation CancelRegistration / TransponderReturned

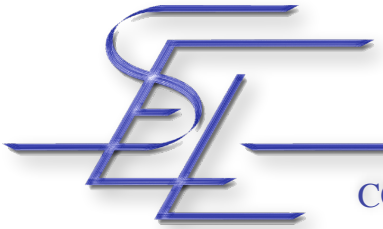
Operation: 407Backend::cancelRegistration(t: Transponder);
Scope: Transponder;
Post:
  not t.active

Operation: 407Backend::transponderReturned(t: Transponder);
Scope: Transponder;
Pre: **not** t.active
Post:
 self.transponder→**excludes**(t)

McGill

Predicate TripCreated

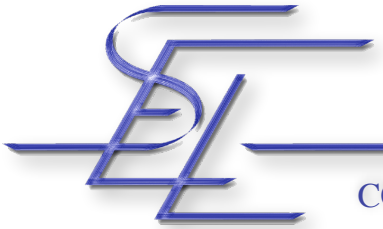
Predicate: 407Backend::TripCreated (v: Vehicle, newTrip: Trip,
when: Time, where: Gantry, class: VehicleClass, kind:);
Body:
newTrip.**oclIsNew**() &
newTrip.detClass = class &
if where.type = GantryKind::entry **then**
newTrip.entryTime = when &
newTrip.entryGantry = where
else
newTrip.exitTime = when &
newTrip.exitGantry = where
end if &
v.trip = v.trip**@pre→append**(newTrip)



McGill

Operation RegisteredPass (1)

Operation: 407Backend::registeredPass
  (t: Transponder, when: Time, class: VehicleClass, pict: Picture);
Scope: Gantry, Vehicle, Trip, Transponder;
Alias:
  myGantry: Gantry = **sender**.gantry;
  v: Vehicle = t.vehicle;
New: newTrip: Trip;
Post:
  t.active &
  if myGantry.type = GantryKind:entry **then**
    self.TripCreated(v, newTrip, when, myGantry, class) &
    if pict.isRecognizable() **then**
      newTrip.detLP = pict.getLicencePlate()
    else
      -- if we can't recognize the licence plate, then don't bother showing it to an operator
      newTrip.detLP = v.plate
    endif



McGill

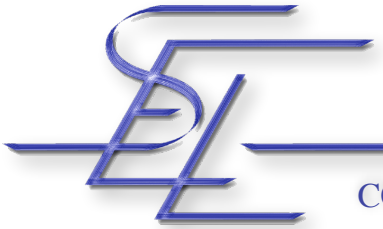
Operation RegisteredPass (2)

Post (continued):

```
  else
    -- it is an exit
    let myLastTrip: Trip = v.trip→last() in
      if myLastTrip.exitGantry→isEmpty() then
        myLastTrip.exitTime = when &
        myLastTrip.exitGantry = myGantry
      else
        -- entry was not registered because of a failure
        self.TripCreated(v, newTrip, when, myGantry, class)
      endif
    endlet
  endif
```

Operation UnregisteredPass (1)

Operation: 407Backend::unregisteredPass(when: Time, class: VehicleClass, pict: Picture);
Scope: Gantry, Vehicle, Trip, Terminal;
Messages: OperatorTerminal::{DisplayPicture}, GovernmentComputer::{GetVehicleInfo}
Alias:
  myGantry: Gantry = sender.gantry;
New:
  newTrip: Trip;
  newVehicle: Vehicle;
Post:
  if myGantry.type = GantryKind:entry **then**
    newTrip.**oclIsNew**() &
    newTrip.entryTime = when &
    newTrip.detClass = class &
    newTrip.entryGantry = myGantry &



McGill

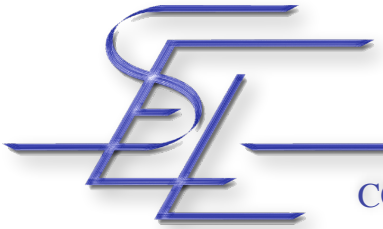
Operation UnregisteredPass (2)

```
Post (continued):
  if pict.isRecognizable() then
    newTrip.detLP = pict.getLicencePlate() &
    let v: Vehicle = self.vehicle→select(plate = lp) in
      if v.oclIsUndefined() then
        newVehicle.oclIsNew() &
        newVehicle.trip = Sequence{newTrip} &
        self.governmentComputer^getVehicleInfo(pict.getLicencePlate())
      else
        v.trip = v.trip@pre→append(newTrip)
      endif
    endlet
  else
    -- license plate is not recognizable, need to display picture on a terminal
    let chosenTerminal: Terminal = self.terminal→any(t | t.registering→isEmpty()) in
      chosenTerminal^displayPicture(newTrip, pict) &
      newTrip.displayedAt = chosenTerminal
    endlet
  endif
```

McGill

Operation UnregisteredPass (3)

```
Post (continued):
  else
    -- it is an exit
    if pict.isRecognizable() then
      let v: Vehicle = self.vehicle→select(plate = pict.getLicencePlate()) in
       if v.oclIsUndefined() then
          newVehicle.oclIsNew() &
          newTrip.oclIsNew() &
          newTrip.exitTime = when &
          newTrip.detClass = class &
          newTrip.exitGantry = myGantry &
          newVehicle.trip = Sequence{newTrip} &
          self.governmentComputer^getVehicleInfo(pict.getLicencePlate())
       else
          let myTrip: Trip = v.trip→last()) in
            if myTrip.exitGantry.oclIsUndefined() then
              myTrip.exitTime = when &
              myTrip.exitGantry = myGantry
            else
              self.TripCreated(v, newTrip, when, myGantry, class) &
            endif
          endlet
       endif
      endlet
```



McGill

Operation UnregisteredPass (4)

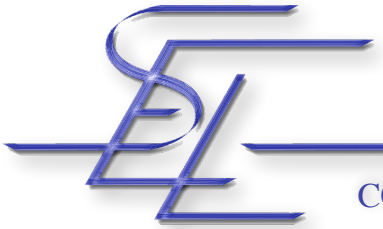
Post (continued):

```
  else
    -- picture unrecognizable
    newTrip.oclIsNew() &
    newTrip.exitTime = when &
    newTrip.exitGantry = myGantry &
    let chosenTerminal: Terminal = self.terminal→any(t |
    t.registering→isEmpty()) in
      chosenTerminal^displayPicture(newTrip, pict) &
      newTrip.displayedAt = chosenTerminal
    endlet
  endif
endif
```

McGill

Operation VehicleInfo (1)

Operation: 407Backend::vehicleInfo
(lp: LicencePlate, class: VehicleClass, ownerName: String, ownerAddress: String);
Scope: Vehicle, Person, Transponder, Account, Terminal;
Messages: PostalService::{DeliverTransponder}, OperatorTerminal::{RegistrationResult}
Alias: registeringVehicle: Vehicle = **self**.terminal.registering**@pre→any**(plate = lp);
New:
newPerson: Person;
Post:
if registeringVehicle.**oclIsUndefined**() **then**
-- vehicleInfo for an unregistered vehicle that takes the highway for the first time
let v: Vehicle = **self**.vehicle**→any**(plate = lp),
owner: Person = **self**.person**→any**(name = ownerName **and** address = ownerAddress) **in**
v.class = class &
if owner.**oclIsUndefined**() **then**
newPerson.**oclIsNew**(name ⇒ ownerName, address ⇒ ownerAddress} &
newPerson.vehicle = Set{v}
else
owner.vehicle**→includes**(v)
endif
endlet

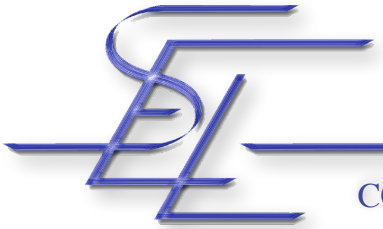


McGill

Operation VehicleInfo (2)

Post (continued):

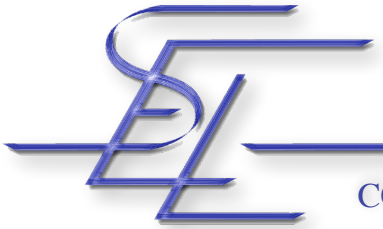
```
  else
    -- vehicleInfo is for a vehicle that is currently being processed for registration by an
    operator
    let success: Boolean = registeringVehicle.class = class and
      registeringVehicle.owner.name = ownerName and
      registeringVehicle.owner.address = ownerAddress in
       if not success then
         -- not specified in the use cases what should happen in this case!
       else
         self.postalService^deliverTransponder(registeringVehicle.owner)
       endif
       registeringVehicle.registeringAt@pre.operatorTerminal^registrationResult(success) &
       registeringVehicle.registeringAt→isEmpty()
    endlet
  endif
```



McGill

Operation RecognizedPlate

```
Operation: 407Backend::recognizedPlate(tr: Trip, lp: LicencePlate);
Scope: Gantry, Vehicle, Trip, Terminal;
Messages: GovernmentComputer::{GetVehicleInfo}
Alias:
  myTerminal: Terminal = sender.terminal;
  v: Vehicle = self.vehicle→select(plate = lp);
New:
  newVehicle: Vehicle;
Post:
  if v.oclIsUndefined() then  -- the vehicle is not known, so this is its first trip
    newVehicle.oclIsNew(plate = lp) &
    newVehicle.trip = Sequence{tr} &
    newVehicle.trip.detLP = lp &
    self.governmentComputer^getVehicleInfo(lp)
  else if tr.exitGantry→isEmpty() then  -- it is an entry
    v.trip = v.trip@pre→append(tr)
  else if v.trip→last().exitGantry@pre→isEmpty() then  -- it is an exit, and the previous trip is incomplete
    if tr.exitTime - v.trip→last().entryTime < 1 then -- we assume that this exit belongs to the previous trip, since not a lot of time has passed since entry
      v.trip→last().exitGantry = tr.exitGantry &
      v.trip→last().exitTime = tr.exitTime &
      self.trip→excludes(tr)
    else  -- since a lot of time has passed since the previous entry, we assume that there was an exit failure for the last trip, and an entry failure for this one
      v.trip = v.trip@pre→append(tr)
    endif
  else  -- it is an exit, and the previous trip is complete
    v.trip = v.trip@pre→append(tr)
  endif &
  myTerminal.displaying→excludes(tr)
```

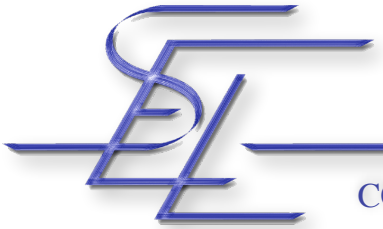


McGill

Calculating Cost of Trip (1)

```
Context: Trip
Alias:
  peak: Boolean = self.entryTime.isPeakHour() or self.exitTime.isPeakHour();
  travelledDistance: Distance =
    if t.entryGantry→isEmpty() or t.exitGantry→isEmpty() then
      minimumDistance
    else
      t.entryGantry.pos.distance(t.exitGantry.pos)
    endif

Def: costForDistance() : Integer =
  if self.detClass = VehicleClass::light then
    if peak then travelledDistance * 16.25
    else travelledDistance * 15.5 endif
  elseif self.detClass = VehicleClass::heavy_single then
    if peak then travelledDistance * 32.5
    else travelledDistance * 31.5 endif
  else
    if peak then travelledDistance * 48.75
    else travelledDistance * 46.5 endif
  endif
```



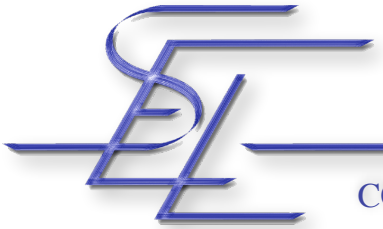
McGill

Calculating Cost of Trip (2)

```
Context: Trip
Def: videoCharge() : Integer =
  if self.vehicle.myTransponder→isEmpty() then 350
  else 0
  endif

Context: Trip
Def: cheatingFine() : Integer =
  if ((self.vehicle.myTransponder→notEmpty() and (self.vehicle.plate <> self.detLP)) or
    ((self.detClass <> VehicleClass::single) and (self.vehicle.myTransponder→isEmpty()) then
    5000
  else 0
  endif

Context: Trip
Def: cost() : Integer =
  self.costForDistance() + self.videoCharge() + self.cheatingFine();
```

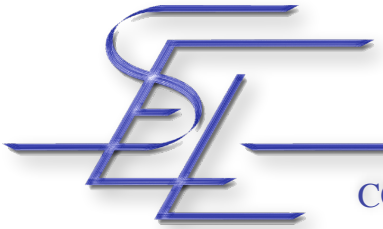


McGill

Operation EndOfMonth (1)

Operation: 407Backend::endOfMonth();
Scope: Account, Person, Vehicle, Trip, Cheque, CreditCard, CreditCardKind;
Messages:
PostalService::{Bill}, GovernmentComputer::{DenyRenewal},
CreditCardCompany::{creditAmount}
Post:

```
-- the bills are sent to registered accounts
self.account→forAll(a |
  let contents: Set(Transaction) in
    contents→includes(Tuple{TransactionKind::monthlyFee, 'Monthly Transponder Fees', 300 *
    a.registeredT→select(active)→size()}) &
    a.registeredT.registeredVehicle.trip→select(t | t.cost() > t.amountPaid)→forAll(t |
      contents→includes(Tuple{TransactionKind::registeredTrip, 'Trip from ... to ...', t.cost() -
      t.amountPaid})) &
  self.postalService^bill(a.owner, contents)
  endlet ) &
```

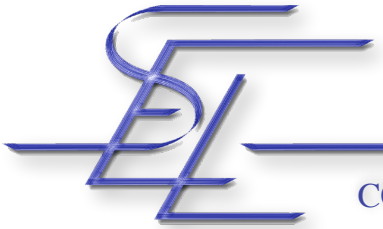


McGill

Operation EndOfMonth (2)

Post (continued):

```
  -- the bills are sent to unregistered vehicle owners
  self.vehicle→forAll(v | v.myTransponder→isEmpty() implies
    let contents: Set(Transaction) in
      v.trip→select(t | t.cost() > t.amountPaid)→forAll(t |
        contents→includes(Tuple{TransactionKind::unregisteredTrip,
        'Trip from ... to ...', t.cost() - t.amountPaid})) &
      self.postalService^bill(v.owner, contents)
    endlet ) &
```

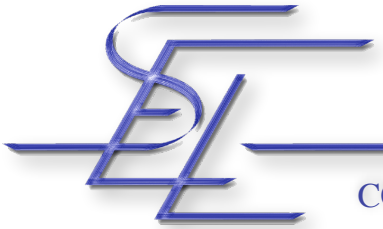


McGill

Operation EndOfMonth (3)

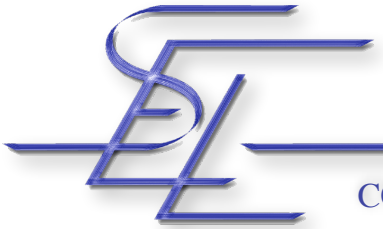
Post (continued):

```
-- the credit cards associated with accounts are charged
let amountToBeCharged = self.account.registeredT→select(active)→size() * 300 +
self.account→forAll(a |
  a.registeredT.registeredVehicle.trip→select(t | t.cost() > t.amountPaid)→collect(cost() -
  amountPaid)→sum() in
  a.billedTo.myKind.creditCardCompany^creditAmount(a.billedTo.number, amount)
endlet &
-- licence plate renewals are denied for customers that are overdue
self.vehicle→forAll(v |
  v.trip→select(t | (t.exitTime < self.clock.now - 3 * Time::months or
    t.entryTime < self.clock.now - 3 * Time::months)→notEmpty() and
    v.owner.currentCheque→isEmpty())
  implies self.governmentComputer^denyRenewal(v.plate))
```



Operation CreditResult

Operation: 407Backend::creditResult(c: CreditCardNumber, amount: Integer, outcome: Boolean);
Scope: CreditCard, CreditCardKind, Account, Person, Vehicle, Trip;
Alias:
  payer: Person = **self**.creditCard→**select**(card | card.number = c).paysFor.owner→**any**()
Messages: PostalService::{PayCreditResult}
Post:
  if outcome **then**
    -- we assume that, if successful, the amount is always the total amount requested
    self.creditCard→**select**(card |
      card.number = c).paysFor.registeredT.registeredVehicle.trip→**forAll**(t |
        t.amountPaid = t.cost())
  endif &
  self.postalSercive^payCreditResult(payer, outcome)



McGill

Operation ChequeReceived

Operation: 407Backend::chequeReceived(p: Person, cheque: ChequeNumber);
Scope: Person, Cheque;
New:
  newCheque: Cheque
Post:
  -- remember that this person has sent a cheque in order to not
  -- deny licence plate renewal at the end of the month
    newCheque.**oclIsNew**(number ⇒ cheque) &
    p.currentCheque = newCheque

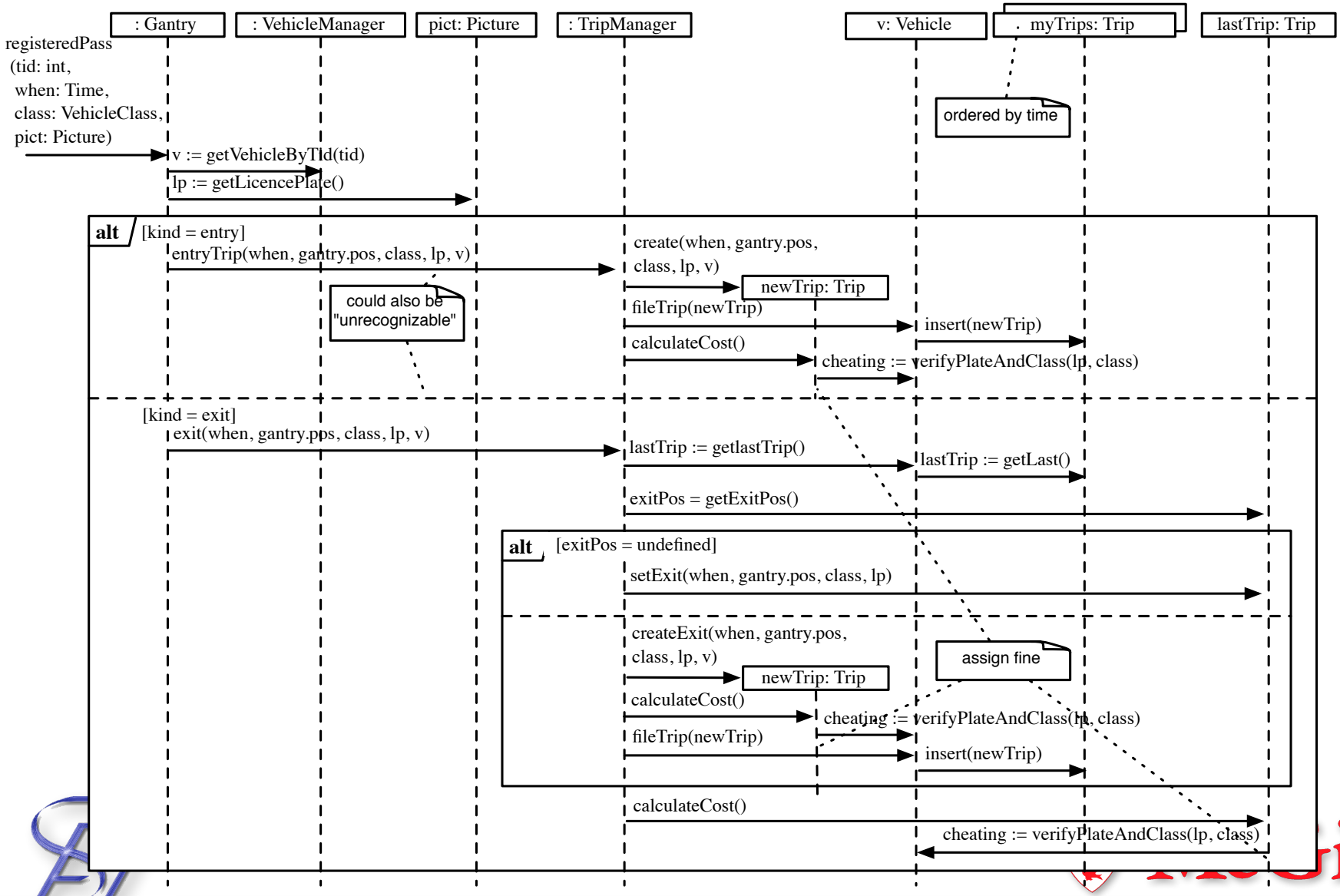
McGill

Operation ChequeResult

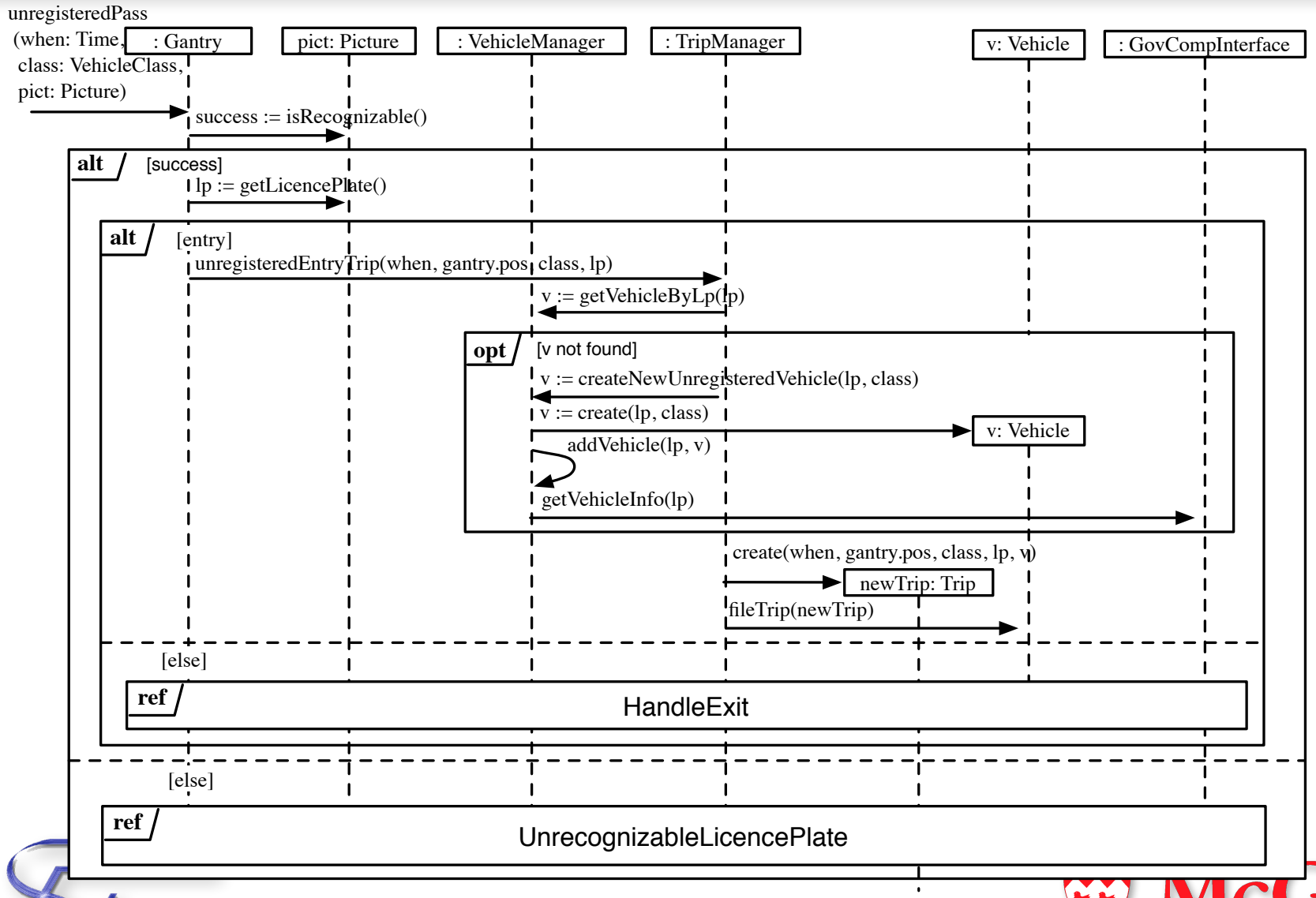
Operation: 407Backend::chequeResult(cheque: ChequeNumber, amount: Integer, outcome: Boolean);
Scope: Trip, Person, Vehicle, Cheque;
Alias:
  payer: Person = **self**.cheque→**any**(ch | ch.number = cheque).sentBy**@pre**;
  trips: Set(Trip) = payer.myVehicle.trip;
Messages: PostalService::{BouncedCheque}
Post:
  if outcome **then**
    trips.amountPaid→**sum**() = trips.amountPaid**@pre**→**sum**() + amount &
    trips→**select**(t | t.amountPaid = t.cost()).exitTime→**max**() < trips→**select**(t | t.amountPaid <
    t.cost()).exitTime→**min**() &
    let oldestNotCompletelyPaidTrip = trips→**any**(t | t.amountPaid < t.cost() **and** t.exitTime <
    trips→**select**(amountPaid < cost().exitTime →**min**()) **in**
      oldestNotCompletelyPaidTrip.cost() - oldestNotCompletelyPaidTrip.amountPaid >
      trips→**select**(t | t.amountPaid < t.cost()).amountPaid→**sum**())
    endlet
  else
    self.postalService^bouncedCheque(payer, cheque)
  endif &
  payer.currentCheque→**isEmpty**()

McGill

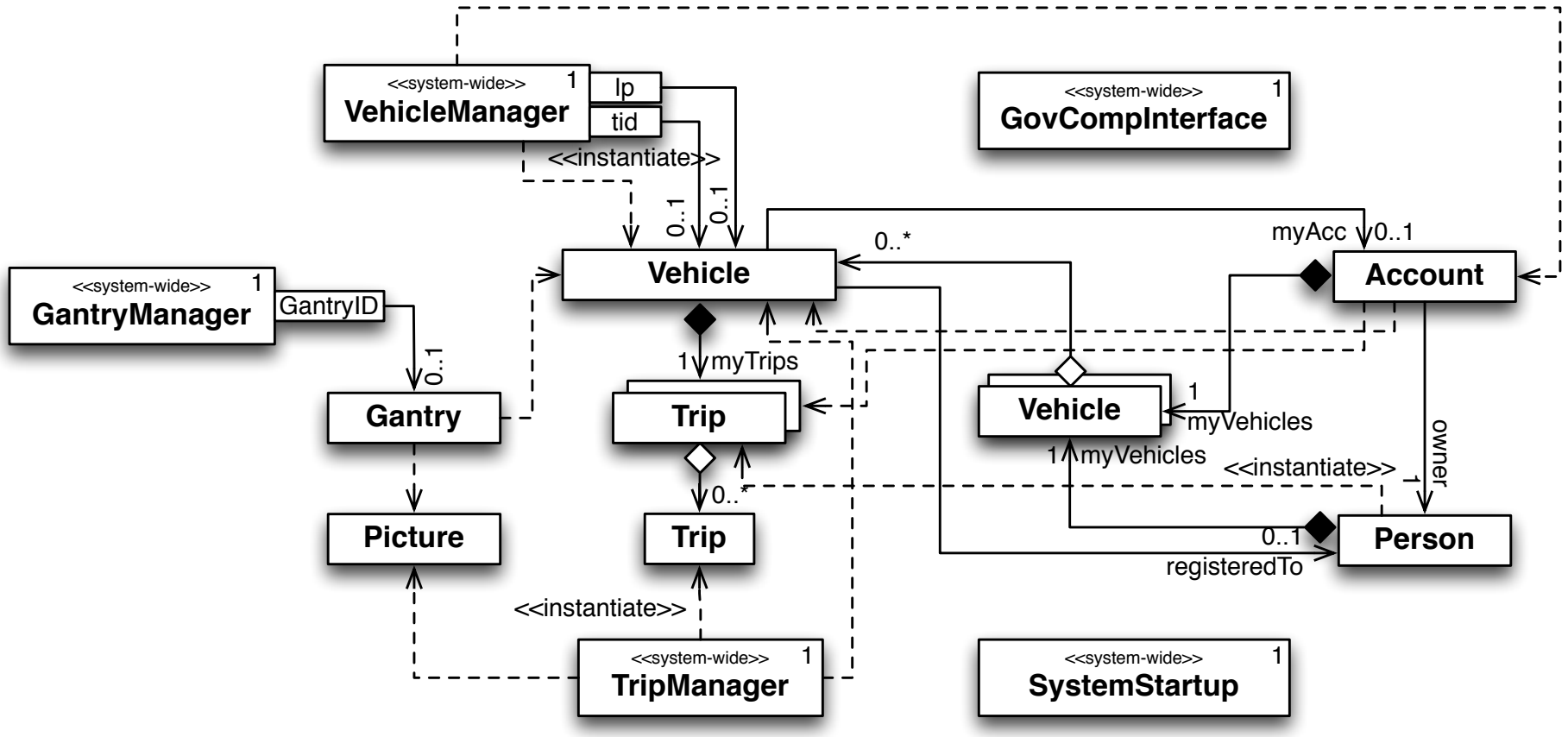
Partial Design of 407 ETR: registeredPass



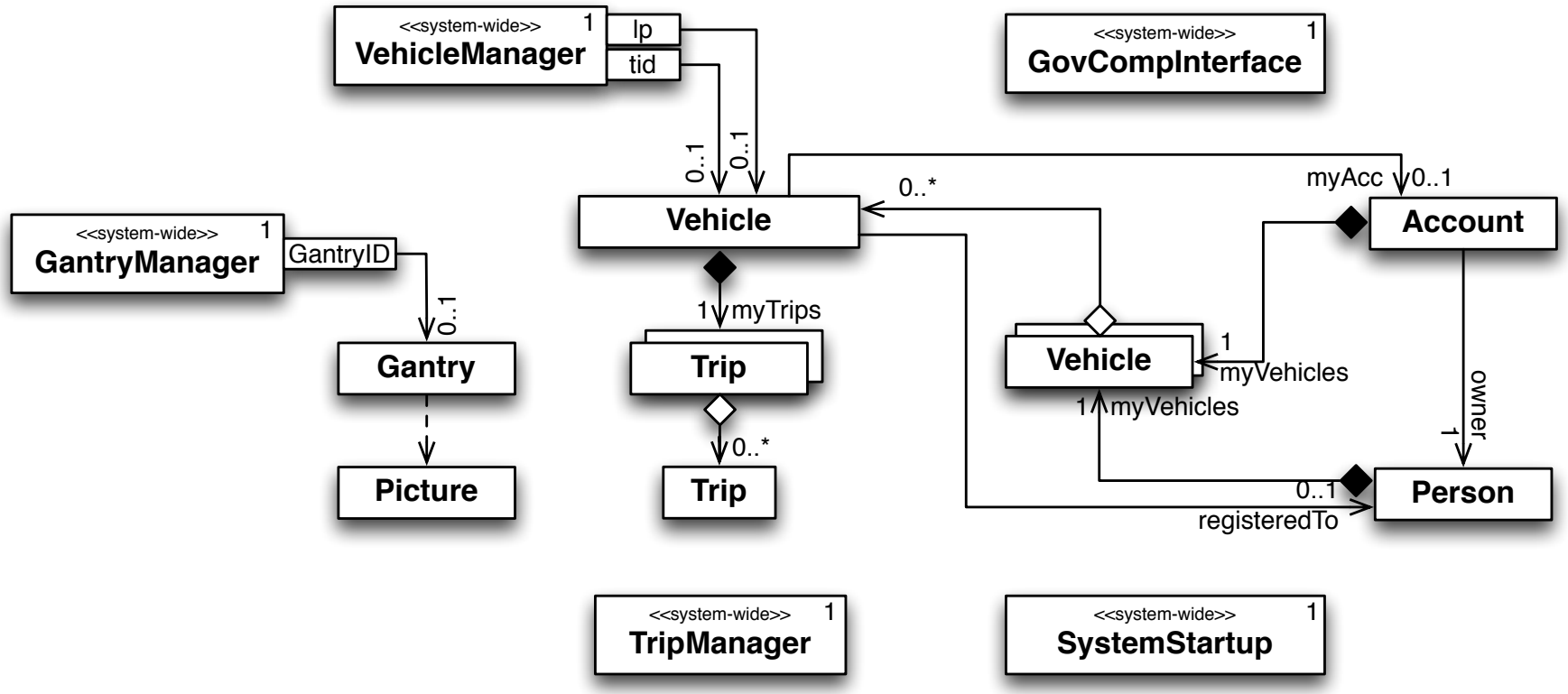
Partial Design of 407 ETR: unregisteredPass



Partial 407 ETR Dependency Diagram



Partial 407 ETR Design Class Diagram (1)



McGill

Partial 407 ETR Design Class Diagram (2)